CONGESTED HIGHWAYS ACTION RESPONSE TEAM
STATE HIGHWAY ADMINISTRATION

Java Benefits and Risk Analysis

Contract DBM-9713-NMS
TSR # 9901961
Document # M361-AR-001R0

May 10, 1999
By
Computer Sciences Corporation and PB Farradyne Inc

Table of Contents

1 Introduction

The Java programming language is being considered for use in development of the CHART II system. Java is being considered based on statements made in the CHART II visioning workshops regarding the desire for a portable user interface. Additionally, prior development team experiences with the Java language demonstrated enhanced programmer productivity due to various features of the language. The Java programming language is relatively new to the software industry. While its use is rapidly expanding, its relative newness requires that a technical assessment be performed to ensure that its use will benefit the CHART II project.

This paper describes benefits of Java and the perceived risks, primarily from a programmer's perspective. It is not intended to be an exhaustive research treatise on Java but rather to common Java issues and CHART II team member specific concerns as well as specific mitigation actions for identified risks. The information has been developed based upon knowledge and experience of the CHART II development team, research performed by them and the early results from feasibility testing.

The alternative to using Java is to code the CHART II system entirely in C++ as was originally planned. Thus much of the discussion, while centered on Java, compares the two approaches.

2 Benefits of Using Java

2.1 Platform Independence

Programs written in Java can be run on any computing platform for which a Java virtual machine is available. Currently, the list of operating systems supporting Java is large, including Windows NT/95/98, Sun Solaris, IBM OS/2, Apple Macintosh, Linux, and HP/UX.

This cross platform capability allows servers written on Java that are originally deployed on a PC to be run on a UNIX server without modification. It also allows a graphical user interface (GUI) to be downloaded from a server and run on a PC or UNIX computer. This portability allows for more options for future growth and makes the CHART II GUI accessible to more end users without dictating a computing platform.

Programs written in C++ target a specific platform. Programs that must run on different platforms require separate build processes, a separate code base and/or code that is conditionally included depending on the platform. This adds a level of complexity to any C++ program that targets multiple platforms.

2.2 Programmer Productivity

2.2.1 Automatic Memory Management / Garbage Collection

One of the greatest benefits of Java from the programmer's perspective is freedom from memory management. It has been estimated that thirty percent of application development and maintenance time with languages such as C++ is spent on memory management issues. Java uses garbage collection to clean up memory automatically when an object is no longer being used.

Freedom from dealing with memory management allows object oriented designs to be implemented much easier than with other languages such as C++. Objects may be passed around the system without worrying who is responsible for freeing memory when the object is no longer needed in the system. This eliminates common programming problems such as memory leaks and illegal memory access (which is a common cause for program crashes).

2.2.2 Syntax

Java's syntax is very close to that of C++ but leaves out the more arcane syntax. This allows a programmer looking at a Java program to have little question as to what a particular piece of code does. In addition, since much of Java's syntax is the same as C++, a C++ programmer can easily understand the code as well.

2.2.3 Object Oriented Language from the Start

Because Java was designed as an object oriented programming language from the start, many built in classes make the programmer's job easier.

2.2.3.1 Threads

The Java language contains a Thread class that makes multi-threaded programming easier to accomplish. This allows a single object to contain both the thread function and the data on which the thread function will operate.

In contrast, C++ programs rely on operating system calls for threading. Thread functions are global functions and are typically passed an instance of a class on which to operate. It is difficult in a C++ program to coordinate the life of the thread function and the objects on which they operate. Mistakes in this area in C++ programs frequently lead to program crashes.

2.2.3.2 Synchronization

The Java language contains built-in synchronization mechanisms for use in multi-threaded programs. These built-in mechanisms make it easy to protect data that is being accessed from multiple threads and to coordinate tasks which involve multiple threads.

In contrast, C++ programs rely on operating system calls to synchronize access to data and coordinate threads. Mistakes in this area may lead to unpredictable program behavior and program crashes.

2.2.3.3 Strings

Java contains a built in String class that allows for manipulation of character strings. The C++ language only recently included a string class, thus many programs rely on operating system specific implementations of a string class, such as the Microsoft MFC CString. This means that there are many different implementations for String manipulation which leads to requiring many conversions to be done in C++ programs between C++ (standard template library) strings and other string implementations. This is especially true when programming graphical user interfaces using the Microsoft MFC library that was used for the CHART II GUI prototype.

2.2.3.4 Arrays

Java contains a built in type for arrays. Any type has an associated array type. An array may therefore be passed as a single object. In C++, passing arrays around in the code requires passing an additional variable to tell how big the array is. This makes it difficult to write functions that return arrays, because you are limited to one return value. Errors in properly coordinating the array size and array lead to program crashes.

2.2.3.5 Run time type checking

In an object oriented program, it is often necessary to determine what class of object you are dealing with in order to perform processing specific to that class of object. All Java objects contain a method that allows you to check the class of the object. Although the C++ language currently supports this through the standard library, it is a compile time option that can lead to program crashes when compile mismatches occur.

2.2.3.6 Interfaces

The Java language does not allow multiple inheritance. Multiple inheritance can complicate code and require knowledge about the internals of the language to be able to predict the results of some code.

Instead, the Java language includes the concept of an Interface. An interface is just what its name implies. It defines the interface required to utilize the functionality of an object. This allows for loosely coupled designs and code that lends itself well to changes that don't ripple throughout the system.

While the C++ language doesn't have explicit support for the "interface" concept, it could be implemented by using pure virtual classes and multiple inheritance. This is more complex than using the built in Java interfaces.

2.2.3.7 Exceptions

Java and C++ allow the use of exception handling to handle the "exceptional" cases in a program. Use of exceptions allows the main stream program flow to be straightforward and easy to understand. Typically code without exception handling ends up having very deeply nested if statements that are very difficult to maintain and are prone to errors.

A problem with exception handling is that if an exception is not handled, it can make the program end unexpectedly. Java helps to avoid this grievous error by enforcing rules at compile time. Every Java method must either handle any exception that can occur within it (or within other classes it uses) or must declare that it throws the exception. When a method declares that it throws an exception, this requires the calling method to either catch it or declare that *it* throws the exception. This requirement makes it easier for a Java program to insure that it will properly handle exceptions. In contrast, C++ does not enforce exception handling and it is easy to leave many unhandled exceptions that will not be discovered until the exceptional condition occurs (which is usually after deployment).

2.3 CORBA to Java Mapping

CORBA allows for programs written in different languages to share software objects. There are many specifications that define how CORBA is to be accessed from within various programming languages. Given Java's built in reference counting and garbage collection, using CORBA from within Java is easier than using CORBA within C++.

The CORBA mapping for C++ involves accessing objects as pointers or "vars". The C++ programmer is responsible for manipulating reference counts for objects they are accessing throughout their code to prevent memory leaks and crashes. This makes the use of CORBA in C++ programs more error prone than using CORBA in Java programs.

Also, the CORBA to C++ mapping maps CORBA strings to character pointers. This means every time a string is returned in IDL, there is a potential for a memory leak on the client side.

2.4 ORB Pluggability

Java 2 ships with a simple ORB included. To use another vendor's ORB requires minimal effort to change a small number of settings. With C++, changing ORBs requires much more effort, usually because vendors use proprietary cross platform threading libraries. Such threading libraries are not needed in Java because threading is built into the language.

2.5 Pluggable Technology Advances

Since Java code is not compiled to an executable, it is possible to take advantage of emerging technological advances in Java virtual machines without changing the source code. An example of this exists today with the recent announcement of the Sun HotSpot virtual machine. This virtual machine automatically optimizes code that is most heavily used over a period of time, thus doing smart, adaptive optimization and increasing performance.

3 Perceived Risks

3.1 Performance

The first thing that comes to mind when someone mentions Java is "it is too slow". This perception usually either comes from information on early releases, knowledge that Java is an interpreted language or because the person has accessed web sites that contain Java applets.

Since Java is interpreted, much of the performance is dependent on the virtual machine on which it runs. New technologies are being applied to virtual machines to make Java performance approach the speed of C++. Examples are the Symantic Just In Time (JIT) compiler that compiles Java code to native code as it executes, or the Sun HotSpot virtual machine, that optimizes the portion of Java code that gets executed the most in a given program.

Part of the cause of slow performance with Java applets is that each time a web page is accessed that includes an applet, the browser downloads the applet code to run in the browser's Java virtual machine. This download time often brings one to associate "slowness" with Java. There have been advances made in lessening download time in the way of local caches for applets that have been accessed previously, and the use of JAR files to compress the Java code and require only a single download. However an approach that requires downloading of applets is unlikely to be used for CHART II.

3.1.1 Risk Mitigation

To mitigate the risk of Java performance becoming an issue, the development team conducted a performance comparison between Java and C++. Evaluation of the results showed that C++ was faster in performing computations, but Java was faster in code that used object oriented features. This paper is attached for reference.

Also, during the ORB evaluation for CHART II, performance tests on the ORBs were conducted. The tests included both C++ and Java ORBs that implemented the same functionality. Although some C++ ORBs were slightly faster than ORBs written in Java, some Java ORBs were faster than C++ ORBs.

On the GUI side, ongoing feasibility testing is demonstrating that Java GUIs draw as fast (or faster) than C++ GUIs that perform the same functionality. Also, preliminary tests with drawing maps show that a Java map can easily meet the 3 second map redraw requirement for CHART II although currently the Java version is slightly slower than the C++ version.

3.2 Tools

Programming tools enhance programmer productivity and as such have become a standard part of any professional programming environment. Many Java development tools exist, however the development team to this date has used only the freely available Java 2 JDK from Sun. The most significant development tool that is currently not available to the team is a graphical debugger. Although having a debugger has not severely constrained development for the feasibility testing, its use will become a necessity when complex application logic is being developed.

Some Java development tools tie themselves to a particular Java version and are slow to update to the rapidly evolving Java products. An example of such a tool is Microsoft Visual J++. Choosing a tool that does not allow a pluggable JDK may render the tool out of date quickly.

3.2.1 Risk Mitigation

The development team has sampled many Java development environments on a trial basis. Although an exhaustive tool evaluation has not been performed, Symantec's Visual Café appears to have the productivity enhancing features that are needed. This product includes an interactive development environment with a full-featured debugger. Version 3.0 of this product allows future Java releases to be plugged in. Other candidate products are Jbuilder 3.0 and Visual Age 3.0.

3.3 Training

Given a staff of C and C++ programmers, there will be some learning curve to become proficient in Java programming. It may be possible that programmers develop code incorrectly due to lack of familiarity with the Java language.

3.3.1 Risk Mitigation

Given Java's similarity to C++ and duplication of much of the C++ syntax, the development team has found the learning curve for C++ object oriented programmers to be small. This learning curve is smaller for server programmers than it is for GUI programmers due to the need for GUI programmers to learn the Java Foundation Class (JFC) and/or the Advanced Windowing Toolkit (AWT) classes.

Even so, the GUI learning curve for small due to the object oriented design of the GUI classes and no imposition of an application framework. The Java GUI classes are intuitive, extensible, and fully within the developer's control. In contrast, the Microsoft Foundation Classes (MFC) that are used under C++ impose an application framework that is not intuitive and often requires many tricks to get it to achieve the desired effect.

A free Java tutorial exists on the Sun web site, and dozens of books are available to help one learn Java. If a person responds better to classroom based training, there are many available Java classes from beginner to expert at local technical training centers. Development team staff at PBFI have Java classroom training available and several team members have taken advantage of this training.

3.4 Show Stopper Bug

It is often feared that a bug so severe will be found that there is no workaround. I this occurs, the Java solution will be rendered infeasible, causing a re-write in C++ which would incur significant additional costs and delays for the project.

3.4.1 Risk Mitigation

First it is worth noting that such risks are inherent in any technical endeavor especially software development which takes place in a complex and rapidly changing technical environment. A primary mitigation method is to ensure that an appropriate methodology is applied for the design

and development processes. Given that the team is following the rigorous Catalyst methodology including extensive use of quality assurance techniques, this risk is somewhat mitigated regardless of the development language used.

As part of feasibility prototyping, the development team has used Java on both GUI and Server applications. Development team members have used Java in previous projects and have yet to encounter such a show stopper. During feasibility testing the team has exercised a Java ORB and Java CORBA services on the project development network. As of this writing tests have been running for weeks, have processed millions of requests, and have yet to demonstrate any of the failures that are typically manifested during endurance testing. Many commercial enterprises, for example Home Depot, have used Java for their internal systems and have not only failed to encounter a show stopper, but they have publicized how good their Java experience has been.

If the worst case occurs and CHART II development somehow, however unlikely, encounters such a show stopper bug and it cannot be fixed and therefore Java must be completely abandoned, there will still exist an object oriented design that can be implemented in C++. In addition, the use of CORBA allows each type of object to be implemented in any programming language. Should Java work fine for one process, i.e., DMS, but not for incident detection, the incident detection server could be written in C++ without losing the DMS code.

In addition, the fact that feasibility testing, as mandated by the Catalyst process, is being conducted for critical CHART II functions will drastically reduce the probability that a major bug will occur.

3.5 Support

As with any programming language, support is supplied through the compiler/tool vendor. There are always risks related to business stability and quality of service for tool vendors. If, for example the team chose to continue to use the freely available Sun JDK as our tool, support is limited unless a support agreement is purchased. Other tools come with an initial support period after which additional support must be purchased.

3.5.1 Risk Mitigation

Purchase a Java development environment acquire support from the vendor, or use the Sun JDK and purchase one of their support options.

3.6 CHART II Map Subsystem

The requirements contained in the CHART II RFP include performance constraints for the map portion of the CHART II GUI, specifically a 3 second maximum redraw time. Feasibility testing has demonstrated that to achieve this performance, there are a number of Java implementation options. Each option has its own benefits and risks.

3.6.1 Option 1 - Use a Java wrapper to OpenGL

OpenGL is a standard graphics specification that has been implemented on many platforms. It was used with C++ for the CHART II GUI prototype. While many Java wrappers exist, there are

two that are suitable for use on CHART II. Of these, one is freely available under the GNU public license and the other is provided by a commercial vendor.

Both wrappers are developed and supported by small companies and there is a risk that support for these products in the future may cease. In fact, the version supplied by the commercial vendor was taken off the market for 3 months due to poor sales, however, it has made a return due to strong reviews.

The freely available wrapper includes source code, which helps to mitigate the risk of the uncertain future of its support, but this implementation does not cover 100% of the OpenGL API. The commercially available wrapper supports 100% of the OpenGL API, however it does not support vector-based fonts. While the need for vector based fonts hasn't been fully established for CHART II and it is likely that alternatives exist, it would be better to not lose this option early on. The commercial version does not include source code. It has a runtime license fee of $25.

In addition to requiring a vendor's Java toolkit, using an OpenGL wrapper will require that the OpenGL libraries exist on any platform where our map will run. This is not a problem for Windows NT and is not believed to be a problem for Solaris, however testing on Solaris has not yet been conducted. In general, using OpenGL may limit the portability of the map subsystem.

3.6.2 Option 2 - Provide a Java wrapper to PB Farradyne Map Library

This map library was used in the CHART II GUI prototype. The map library is tightly coupled to the PB Farradyne MIST product. Regardless of its use in Java or C++, effort is required to de-couple this Map library from the MIST product. The resulting map application would then only run on Windows platforms and the portability of Java would not be realized for this subsystem.

3.6.3 Risk Mitigation

Investigation into the different map options has been done and the development team has identified the strengths and benefits of the options. If it is at some point decided that the CHART II GUI will only run on Windows platforms, then the option with the least risk is to provide a wrapper to the PB Farradyne map library. The effort for doing this would be necessary whether the CHART II GUI is implemented in C++ or Java.

Given that a reason for considering Java is that portability is desirable, the tradeoff between the better implementation of OpenGL wrapper by the commercial vendor must be weighed against the risk of the uncertain future of this vendor. As a fallback position, should the commercial vendor stop supporting their OpenGL wrapper, it would be possible to switch to the GNU wrapper implementation, filling in where its OpenGL mapping is incomplete.

Prototyping using the OpenGL wrappers has demonstrated that using Java and OpenGL for the map interface is feasible. The OpenGL wrappers need to be tested portability to address remaining technical risks. In the worst case, users of a CHART II GUI on a platform that does not have the required graphics libraries would not be able to display the CHART II map. However, they would be able to use the rest of the CHART II application. Investigation of the most likely platforms to be used and their capabilities in regard to OpenGL libraries is necessary to determine the potential impact.

3.7 Operating System Specific Features Availability

Because Java is platform independent, if there is a need to access features available only to a specific operating system, native code would need to be written in C or C++.

One function to which this scenario may apply is Windows NT security. If it is required that the same login that is used for a user's NT system, be used for access to the CHART II application, then the code to allow this cannot be written in Java.

3.7.1 Risk Mitigation

It is questionable as to whether Windows NT security is the right choice for CHART II. Choosing to use Windows NT security would have a number of implications:

The GUI or the server would become platform dependent, depending on how security is implemented.

Due to the distributed nature of the CHART II application, it may be possible for a user to circumvent the application security provided in CHART II if the workstation logon is used to also gain access to CHART II.

Should it be determined that NT security or another operating system specific feature be required, Java provides the Java Native Interface (JNI). JNI allows Java objects to be used in C++ code, and vice versa. Using JNI, code to take advantage of operating system specific features can be written in C++ and accessed from the Java code. Of course the application becomes non-portable, however the portions of code that are non-portable are well defined and can be implemented for each target platform. This is much better than having to port the entire application.

3.8 Security

A concern has been expressed that using Java will require users of a CHART II GUI to enable the Java virtual machine on their browsers, which may not be desirable due to security risks that rogue applets could potentially cause.

3.8.1 Risk Mitigation

Running code as an applet is just one of the options available for implementing the application using Java. If this approach is for some reason required, the Java virtual machine imposes tight security on applets that disallows applets from accessing files on the client machine, opening network connections, and many other actions a program might attempt to perform. If an applet needs to do some of these things, Java contains a way to develop a "Signed Applet." The Signed Applet uses public key cryptography, which allows client machines to control access so that only approved applets, may perform the restricted actions.

Another way to use Java is as a stand-alone application. Using this method, the CHART II Java application would run in the same way as a C++ executable. The Java virtual machine executable and the Java byte code would be distributed to a client machine and the application would run using the Java virtual machine. The Java application would be able to do anything the C++ application would be able to do with regard to accessing the local machine, because the Java virtual machine imposes no restrictions on applications. The security risks exposed by a Java application (as opposed to applet) are no greater than those exposed by a C++ application.

4 Conclusion

With any software development technology, new or old, there is some risk involved due to lack of a complete lexicon of implemented functions and variations in the experience level of the development team. One step in minimizing risk is to gain experience and prove effectiveness through using the technology. Java is no different in this regard, and the same is true for CORBA. The CHART II development team has conducted feasibility tests with Java and with CORBA to determine applicability. In fact, the CHART II prototype GUI is another implementation of feasibility testing of many of the new implementation concepts derived from the CHART II RFP.

The highest risk area for a Java solution exists with the GUI and is in the area of the mapping subsystem. Thus far, feasibility testing has identified viable options for implementation although some risks remain, primarily in the area of support for third party OpenGL wrappers. The tradeoff for assuming this risk is portability of the CHART II GUI map.

The development team believes that the demonstrated benefits of using Java warrant its use for CHART II GUI and Server applications. Through the team's collective experiences, and knowledge gained through feasibility testing, it appears that no issues exist to deter the use of Java and that its use will enhance the team's ability to produce stable and reliable applications.

It is therefore recommended that Java be used for development of the CHART II Release 1 Build 1 application. Pending positive results from the remaining feasibility testing and experience in developing the first release, it should be considered as appropriate for the remainder of the CHART II application development.

Acronyms

API	Application Programming Interface
AWT	Advanced Windowing Toolkit
CHART	Congested Highways Action Response Team
RFP	Request for Proposal
CORBA	Common Object Request Broker Architecture
GUI	Graphical User Interface
JDK	Java Development Kit
JFC	Java Foundation Class
JIT	Just in Time
JNI	Java Native Interface
MFC	Microsoft Foundation Class
ORB	Object Request Broker